Erstellung einer optimierten JAVA-GUI zur Visualisierung und Bearbeitung von Audio

Bachelorarbeit im Studiengang
Informatik und Multimedia

an der Fakultät für Angewandte Informatik der Universität Augsburg

Vorgelegt von

Sebastian Adolf

Februar 2006

unter Betreuung von:
Prof. Dr. Rainer Lienhart
Lehrstuhl Multimedia-Computing
Universität Augsburg

Erstgutachter:
Prof. Dr. Rainer Lienhart
Institut für Informatik
Lehrstuhl für Multimedia-Computing

Zweitgutachter:
Prof. Dr. Elisabeth André
Institut für Informatik
Lehrstuhl für Multimedia-Konzepte und Anwendungen

Danksagung Mein besonderer Dank gilt Prof Dr. Rainer Lienhart für die Ermöglichung und Betreuung dieser Bachelorarbeit.
Desweiteren möchte ich Gregor van den Boogaart für seine hilfreiche Unterstützung beim Programmieren und der Anbindung des Programmes an seinen „Gabor-Core“ danken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der Gehörsinn repräsentiert zusammen mit dem Gesichtsinn die menschlichen „Fernsinne“, die für die Außenwahrnehmung von essentieller Bedeutung sind. Die optische Wahrnehmung tritt allerdings bei vielen Menschen erfahrungsgemäß in wesentlich stärkerer Ausprägung auf, weil akustische Eindrücke lediglich zeitgebunden rezipiert werden können und man deshalb zu ihrer Verarbeitung eine sehr viel höhere Konzentration als zur Analyse zeitlich neutraler, bildlicher Strukturen aufwenden muss. Infolge dessen basiert die Wahrnehmung und produktive Umsetzung vieler Eindrücke und Anforderungen des täglichen (Arbeits-)Lebens primär auf dem Gesichtssinn. Um nun bei vorrangig auditiven Medien die kommunikative Diskrepanz zwischen Gehör- und Gesichtssinn zu verringern und dem jeweiligen Menschen einen intuitiveren und damit einfacheren Umgang mit akustischen Strukturen zu ermöglichen, erscheint es naheliegend, deren „Daten“ und Informationsgehalt auf ein Art zu präsentieren, die vor allem die visuelle Wahrnehmung des Rezipienten anspricht. Durch diese Transformation wird überdies die temporal bedingte 1-Dimensionalität der Akustik kompensiert, indem sie durch eine mehrdimensionale Bildlichkeit repräsentiert und „festgehalten“ wird: das zeitliche Nacheinander wird zu einem räumlichen Nebeneinander. Die im Folgenden vorgestellten Software stellt sich dem Anspruch, diesen Ansatz in die Realität umzusetzen.

1.1 Motivation

Diese Bachelorarbeit hatte sich die Erstellung einer graphischen Benutzeroberfläche (genannt GaborGUI) zum Ziel gesetzt, die es ermöglichen soll, Audiodaten mit den Mitteln der Bildverarbeitung zu manipulieren und zu analysieren. Die Grundlage dieser GUI ist eine native C++ - Bibliothek (GaborCORE) [3], welche sämtliche Berechnungen, Veränderungen und Visualisierungen der Audiodaten vornimmt. Über eine Schnittstelle werden dann die entsprechenden Funktionen und erzeugten Daten der GUI zur Verfügung gestellt, die Kom-

bination beider Elemente trägt den Namen „AudioBrush“[2].
Um einen universellen Einsatz des Programmes zu gewährleisten, sollte es plattformunabhängig, ressourcenschonend und portabel angelegt sein. Darüber hinaus ist ein möglichst modularer Aufbau der Applikation wünschenswert, der auch zukünftigen Anforderungen gewachsen ist.

1.2 „Klassische“ Bearbeitung von Audiodaten

In den meisten Programmen zur Editierung und Visualierung von Audiodaten findet die Bearbeitung lediglich in einem 2-dimensionalem Rahmen statt, nämlich Zeit (x-Achse) sowie Amplitude (y-Achse). Unter diesen Voraussetzungen ist es möglich, vorhandenes Audiomaterial zufriedenstellend in diesen beiden Dimensionen zu bearbeiten. Typische Operationen wären hierbei das Schneiden von Aufnahmen, die Anhebung oder Absenkung der Lautstärke, und ähnliches. Sollen jedoch auf der Grundlage dieser Darstellungsart sensiblere Eingriffe in das Material vorgenommen werden, wie zum Beispiel das Entfernen eines bestimmten (Stör-)Geräusches oder auch das Hervorheben eines bestimmten Klanges, so erweist sich dies durch die aufgrund der 2-Dimensionalität fehlenden Frequenzdarstellung als schlichtweg nicht möglich.
Um dennoch eine Bearbeitung solcher Problemfälle zu ermöglichen, bieten viele dieser Programme den Einsatz von Filtern (oftmals in Form von Plugins) an, welche das Audiosignal auch im Frequenzbereich bearbeiten können. Dieser Ansatz kann zwar durchaus (nicht zuletzt auch aufgrund der recht hohen „Intelligenz“ mancher Filter) zum gewünschten Ergebnis führen, jedoch ist oftmals die Suche nach den exakten Parameter-Einstellungen vergleichsweise mühselig, langwierig und erfordert für eine effektive Verwendung Fingerspitzengefühl und einen hohen Grad an Erfahrung im Umgang mit dem jeweiligen Programm.

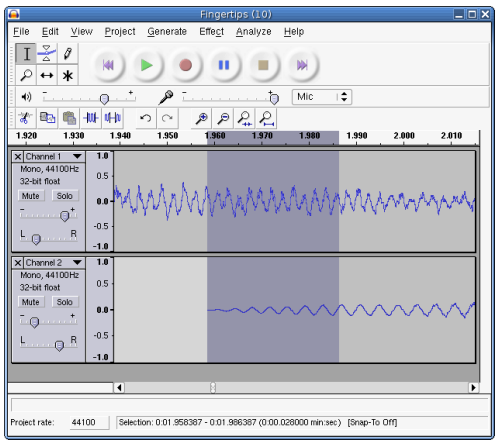


Abbildung 1.1: Klassisches Audio-Editing-Tool (http://audacity.sourceforge.net/)

1.3 Bearbeitung mit AudioBrush

Das Konzept von AudioBrush beschreitet im Gegensatz dazu einen wesentlich benutzerfreundlicheren Weg: die Einführung einer dritten Dimension - der Frequenz. Zur intuitiven Visualisierung dieser drei Komponenten wird auf ein Spektrogramm zurückgegriffen, welches unter Zuhilfenahme einer speziellen Analysemethode (Gabor-Analysis[3]) erzeugt wird. Das Ergebnis ist ein Bild, dessen x-Achse, wie in der klassischen Wellenform-Darstellung auch, die Zeit darstellt und die y-Achse wie gewohnt den Frequenzverlauf. Die dritte Dimension, welche die Amplitude repräsentiert, wird durch die unterschiedliche Farbintensität der einzelnen Frequenzen wiedergegeben (Graustufenwerte von 0 = kein Signal bis 255 = 0db). Wie das Bild dann konkret aussieht, hängt im wesentlichen von zwei Parametern ab :

Samplingfrequenz Diese bestimmt den auswertbaren Frequenzbereich in Abhängigkeit vom Nyquist-Theorem (d.h. der darstellbare Bereich liegt zwischen 0 Hz und der halben Samplingfrequenz, bei 48.0 kHz also von 0 - 24 kHz).

Zeitfenster Dieses Fenster (dessen Größe im vorliegenden Programm durch den Parameter bCrit repräsentiert wird) setzt fest, wie hoch die Auflösung der Frequenzen ist. Ein großes Zeitfenster sorgt zwar für eine sehr feine Auflösung der Frequenzen, verringert aber auch dementsprechend die Auflösung der Zeitachse (und natürlich vice versa).

Betrachtet man das entstandene Bild, so kann man während des synchronen Abspielens der Audiodaten auch ohne große Vorkenntnisse rasch erkennen, wie Ton und Bild zusammenhängen. Man sieht beispielsweise einem Musikstück an, wann der Bass hinzutritt, wie die Gesangsstimme verläuft oder auch in welchem Moment ein Becken angeschlagen wird. Auch die bereits erwähnten Störgeräusche lassen sich meist schnell identifizieren, da sie in den meisten Fällen dem intuitiv als „natürlichen“ empfundenen Verlauf eines Spektrogramms im Wege stehen. Um mit dem nun auf diese Weise optisch vorliegenden Audio-Material zu arbeiten, bedarf es zu allererst der Selektion einer „Region of Interest“ (ROI). Diese kann aus einem simplen Rechteck bestehen (rectangle-selection), aber auch aus einem „offenen“ Polygonzug (inklusive einstellbarer Bandbreite). Darüber hinaus ist eine Kombination aus mehreren verschiedenen Elementen möglich. Da es außerdem gerade im musikalischen Bereich oftmals sinnvoll ist, zu einem bereits ausgewählten „Element“ die dazugehörigen Obertöne zu markieren, können diese auf Wunsch mittels einfachem Knopfdruck der ROI hinzugefügt werden. Ist die gewünschte ROI vollständig erfasst, kann mit der Bearbeitung der Daten begonnen werden. Dazu stehen verschiedenen Operationen zur Verfügung (z.B. das Freistellen oder Löschen der durch die ROI markierten Teile), so dass für eine differenzierte Modifikation der ausgewählten Partien die der jeweiligen Aufgabenstellung am dienlichsten erscheinende Bearbeitungsmethode gewählt und angewendet werden kann. Nachdem der Benutzer alle gewünschten Operationen ausgeführt,

kann er sich ein neues Spektrogramm errechnen lassen, in welchem nun die ausgeführten Änderungen sichtbar werden.

Kapitel 2

Benutzung von AudioBrush

Das folgende Kapitel beschäftigt sich eingehend mit der Installation der Software, der Beschreibung der Oberfläche sowie mit einigen beispielhaften Anwendungsfällen.

2.1 Installation

2.1.1 Systemvorraussetzungen

Zur zuverlässigen Benutzung von AudioBrush müssen folgende Systemvoraussetzungen erfüllt werden :

- ein Betriebssystem, auf welchem das JRE (Java Runtime Enviroment) in der Version 1.5 oder höher verfügbar ist
- einen Arbeitsspeicher von 256MB oder größer
- korrekt installierte OpenGL-Treiber (für die Verwendung des JOGL-Modus)

Desweiteren ist es empfehlenswert, das dem Anwender

- eine Maus mit Scrollrad,
- ein hochauflösendes Display (1280x1024 Pixel oder mehr), sowie
- ein hochwertiges Lautsprechersystem

zur Verfügung stehen.

2.1.2 Kopieren und Ausführen der Programmdateien

Die Installation des Programmes ist in zwei verschiedenen Varianten möglich, die an jeweils unterschiedliche Systemvoraussetzungen angepasst sind:

Lite Enthält lediglich die JAVA-Dateien sowie die eigentliche Library. Diese Variante ist zu empfehlen, falls auf dem System bereits alle notwendigen nativen Bibliotheken installiert sind (Intel ™IPP ™,Intel ™Math Kernel ™[1],OpenCV[2],JOGL-JSR231[3]). Fehlt eine (oder mehrere) dieser Librarys, so sollte auf die Benutzung der zweiten Variante ausgewichen werden.

Full In diesem Package sind sämtliche nativen Bibliotheken enthalten, die zum Betrieb des Programmes notwendig sind. Da sie alle im root-Verzeichnis des Programmes liegen, ist kein Eingriff in das Betriebsystem notwendig.

Beiden Varianten ist gemein,dass sie als gepacktes Archiv vorliegen und einfach nur im gewünschtem Ordner entpackt zu werden brauchen und durch den Aufruf von

```
AudioBrush
```

gestartet werden. Alternativ kann das Programm auch durch

```
java -Djava.ext.dirs=. system.GaborGUIMain
```

innerhalb der Kommandozeile aufgerufen werden.

[1]http://www.intel.com/cd/software/products/asmo-na/eng/perflib/index.htm
[2]http://sourceforge.net/projects/opencvlibrary/
[3]https://jogl.dev.java.net/

2.2 Beschreibung der Benutzeroberfläche

Die Oberfläche von AudioBrush lässt sich in fünf verschiedene Bereiche aufteilen :

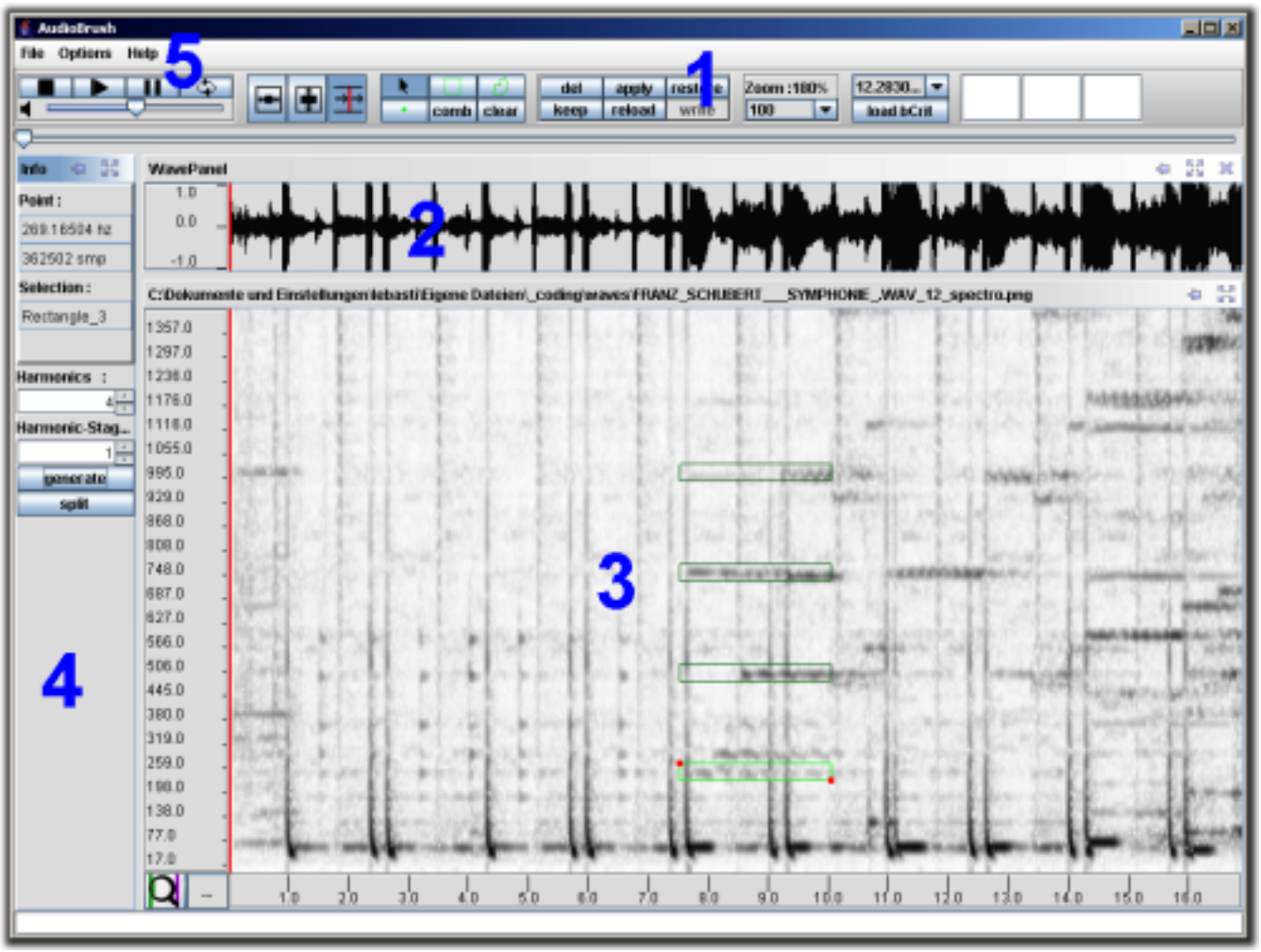


Abbildung 2.1: 1.Werkzeugleiste, 2.Wellenform,3.Spektrogramm, 4.Dynamisches Info-Panel, 5.Menüleiste

2.2.1 Werkzeugleiste

1. Wiedergabe - Hier wird das Abspielen der Audiodatei gesteuert (Start, Stop, Pause), die Abspielposition und die Lautstärke eingestellt sowie der Loop-Modus an- und ausgeschaltet.

2. Ansicht - Man kann die Ansicht des Spektrogrammes auf volle Bildschirmhöhe oder

-breite erweitern sowie festlegen, ob der Bildschirmausschnitt der aktuellen Wiedergabeposition angepasst werden soll.

3. Selektionen - Es stehen folgende Möglichkeiten zu Auswahl : Markierung eines Rechteckes, eines offenen Polygons, eines Punktes, das Hinzufügen der Harmonischen zu einer Selektion (Combing) sowie das Löschen aller Selektionen (Clear).

4. Bearbeitung - Eine Selektion kann entweder gelöscht werden (del) oder freigestellt werden (keep). Durch Drücken von „apply“ wird die Operation angewandt; “reload“ berechnet das neue Spektrogramm und zeigt es an, mit „restore“ wird dagegen die Ursprungsdatei wieder hergestellt.

5. Zeitfenstereinstellung - Hier wird die Größe des gewünschten Zeitfensters (bCrit) angezeigt und gegebenenfalls neu gewählt. „load bCrit“ lädt dabei direkt die aktuelle Datei mit dem neuen bCrit-Wert.

Abbildung 2.2: Screenshot Toolbar

2.2.2 Wellenform-Ansicht

Darstellung der Audiodaten in 2-dimensionaler Form. Die x-Achse stellt den Zeitverlauf dar, die y-Achse die Stärke des Signals (Amplitude). Diese Ansicht passt sich automatisch dem gewähltem Ausschnitt des Spektrogrammes sowie der Höhe des Fensters an.

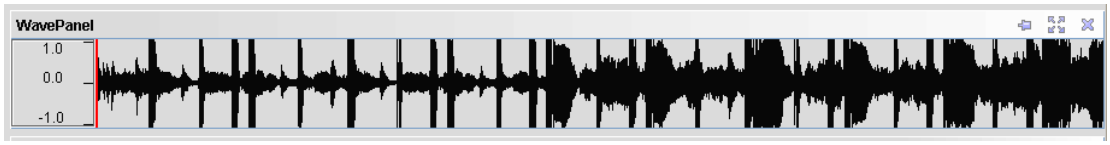


Abbildung 2.3: Screenshot Welleform-Darstellung

2.2.3 Spektrogramm-Ansicht

In dieser Ansicht werden sämtliche Bearbeitungsfunktionen ausgeführt. Die x-Achse repräsentiert die Zeit, die y-Achse die Frequenz. Die Stärke des Signals wird durch die verschiedenen Graustufen veranschaulicht. Der rote Marker zeigt die aktuelle Abspielposition an, der grüne und der blaue Marker stellen die Grenzen des Loops dar (welche mit der Maus verschoben werden können). Der „Lupen"-Knopf verschiebt und zoomt die Ansicht so, das genau der Bereich zwischen diesen beiden Lokatoren angezeigt wird.

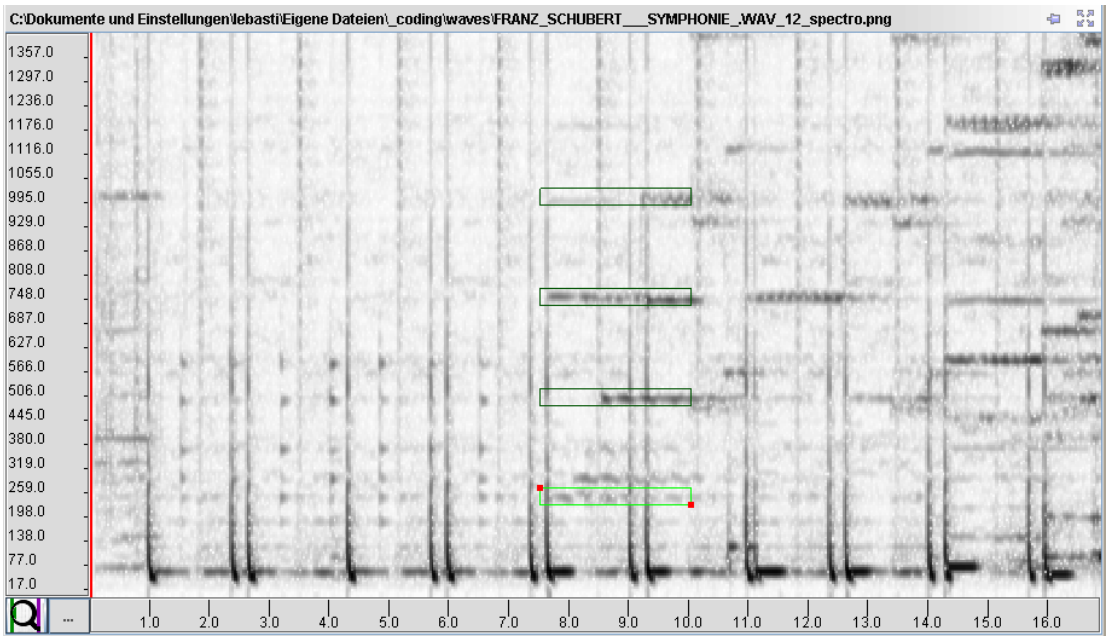
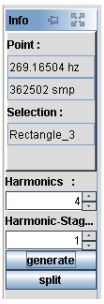


Abbildung 2.4: Screenshot Spektrogrammdarstellung

2.2.4 Dynamisches Info-Panel



Dieses Panel enthält kontextsensitive Informationen und Aktionen zum aktuell ausgewählten Punkt und der dazugehörigen Selektion. Unter anderem können der Name der Selektion, die Frequenz des markierten Punktes, die Bandbreite eines offenen Polygons oder auch die Anzahl der Obertöne (Harmonics) einer „Comb"-Selektion angezeigt werden.

2.2.5 Menüleiste

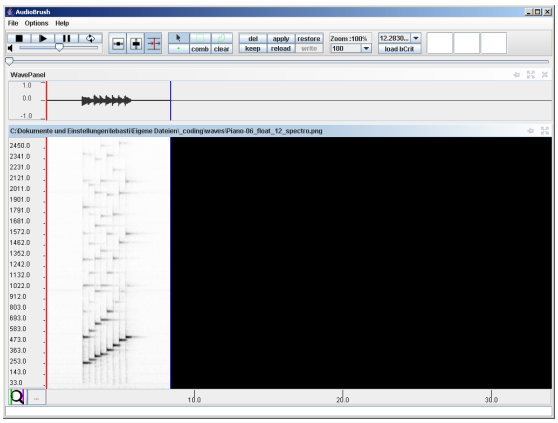
File „Open“ öffnet eine Datei (Format Wave, 32bit, 48khz, Mono) mit dem bCrit-Wert, der in der Werkzeugleiste angezeigt wird, Open(bCrit) lässt den Benutzer den bCrit-Wert per Menü beim Öffnen auswählen. Neben dem Schließen einer Datei und dem Beenden des Programmes sind hier auch die letzten zehn Dateien zum schnellen Laden aufgeführt.

Options Unter „Preferences“ werden die Einstellung der Sound- sowie der Grafikausgabe (JOGL/Java2D) vorgenommen.

2.3 Bearbeiten von Audiodateien

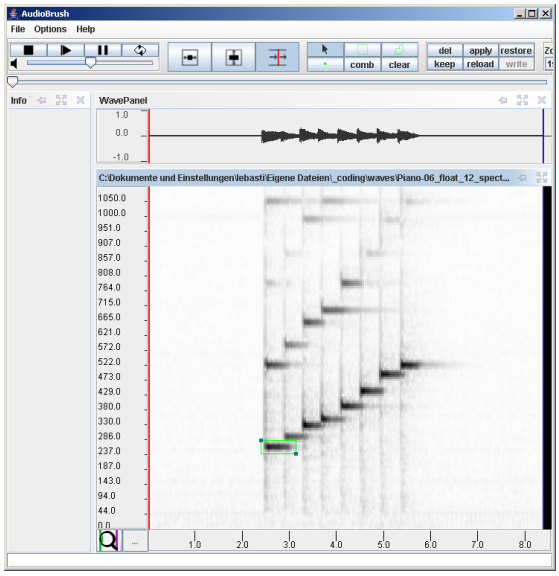
Am Beispiel einer einfachen Klavieraufnahme (`Piano-06_float.wav`, enthalten auf dem Datenträger) wird eine Schritt-für-Schritt-Anleitung zur Benutzung von AudioBrush gegeben :

Laden der Audiodatei



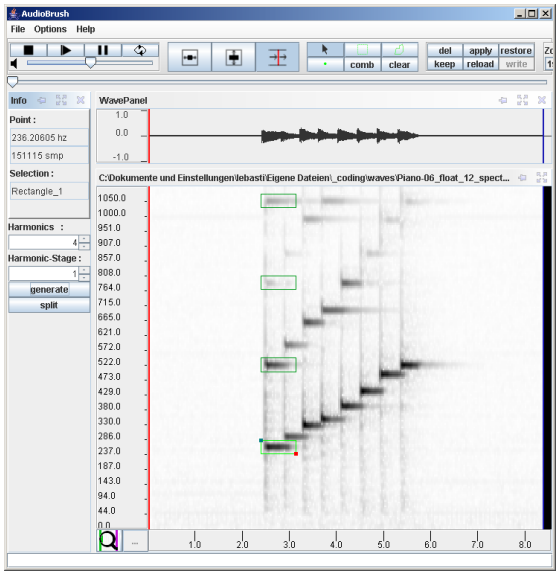
Zunächst lädt der Benutzer die Datei `Piano-06_float.wav` über File-Load(bCrit) mit dem bCrit-Wert 12.2830.

Markierung einer „Region of Interest“



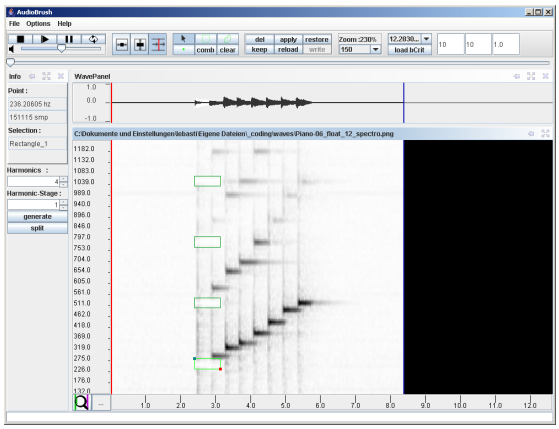
Anschliessend zoomt man die Ansicht auf ca. 230% und markiert mit dem „Rectangle-Tool“ einen der Töne, die durch die horizontal verlaufenden schwarzen „Balken“ repräsentiert werden.

Erstellen einer Kamm-Selektion



Der nächste Schritt ist die Erstellung einer „comb“-Selektion. Hierzu markiert man mit dem „Pfeil“-Tool das erstellte Rechteck und drückt auf den „comb“-Knopf. Im „Dynamic-Info-Panel“ erscheint nun die Möglichkeit, die Anzahl der Obertöne und die Position des Ursprungsrechtecks in der Obertonreihe festzulegen. In diesem Falle sollen vier Obertöne markiert werden, der Ursprung ist der erste Ton der Reihe.

Entfernen der ausgewählten Elemente



In diesem Beispiel sollen nun die selektierten Daten aus der Datei gelöscht werden. Hierzu wird in der Werkzeugleiste der „del“-Knopf betätigt, welcher den Löschmodus aktiviert. Durch drücken von „apply“ wird die Operation ausgeführt. Während der Dauer des Vorgangs ist das GUI gespeert (der Knopf bleibt „gedrückt“). Durch einen klick auf „reload“ werden die Änderungen sichtbar gemacht. Fällt das Ergebnis nicht wie gewünscht aus, so kann man durch den „restore“-Knopf die ursprüngliche Version wiederherstellen.

Kapitel 3

Entwurf und Implementierung

Dieses Kapitel widmet sich zuerst dem abstrakten Aufbau des Programmes. Im Anschluss daran wird die konkrete Implementierung erläutert und den Abschluss bildet ein Abschnitt über die Einbindung der nativen Programmteile in den Java-Code.

3.1 Programmentwurf

Aufgrund der Objektorientierung der verwendeten Programmiersprache JAVA folgt das Programmdesign einem modularen Ansatz unter Berücksichtigung typischer OOP-Techniken. Die Basis der eigentlichen Anzeige bildet die Hauptfensterklasse `GaborGUIFrame`. In ihr werden die einzelnen Anzeigemodule zusammengeführt und verwaltet. Die Kommunikation zwischen ihren Teilen sowie der Audiowiedergabe erfolgt über eine Singletonklasse `EventTransfer`. Sie stellt Referenzen zu den Event-Klassen `GaborEvents`, dem Audiowiedergabemodul `BasicPlayer` und der Hauptfensterklasse bereit.
Da das Programm zwei unterschiedliche Arten der Bildanzeige beherrschen soll, wird diese über eine Abstraktionsebene angesprochen. Ein Interface (`RendererInterface`) und eine abstrakte Klasse (`AbstractGaborPanel`) sorgen dabei für die Vereinheitlichung der Anzeigemodule Java2D/JOGL. Für die Synchronisation der einzelnen Elemente ist die Klasse `GUIUpdateThread` zuständig. Sie setzt die Position der Abspielanzeige (`PlayMarker`, `AudioSlider`) und initiiert das Neuzeichnen der geänderten Elemente. Damit es dem Anwender möglich ist, bestimmte Aktionen wie das Starten/Pausieren der Wiedergabe unabhängig vom aktuellen Fokus auszulösen, musste der Event-Dispatch-Thread zur Verarbeitung von Tastatur-Ereignissen übergangen werden. Diesen Part übernimmt die Klasse `GlobalKeyListener`.
Die Verbindung „Java - C++“ wird durch die Klassen des packages `nativeinterface` hergestellt. Die Übergabe der eigentlich Bearbeitungsfunktionen erfolgt durch die Klasse `SelectionLayerManager` und der entsprechenden Methode der Klasse `JNIEdit`.

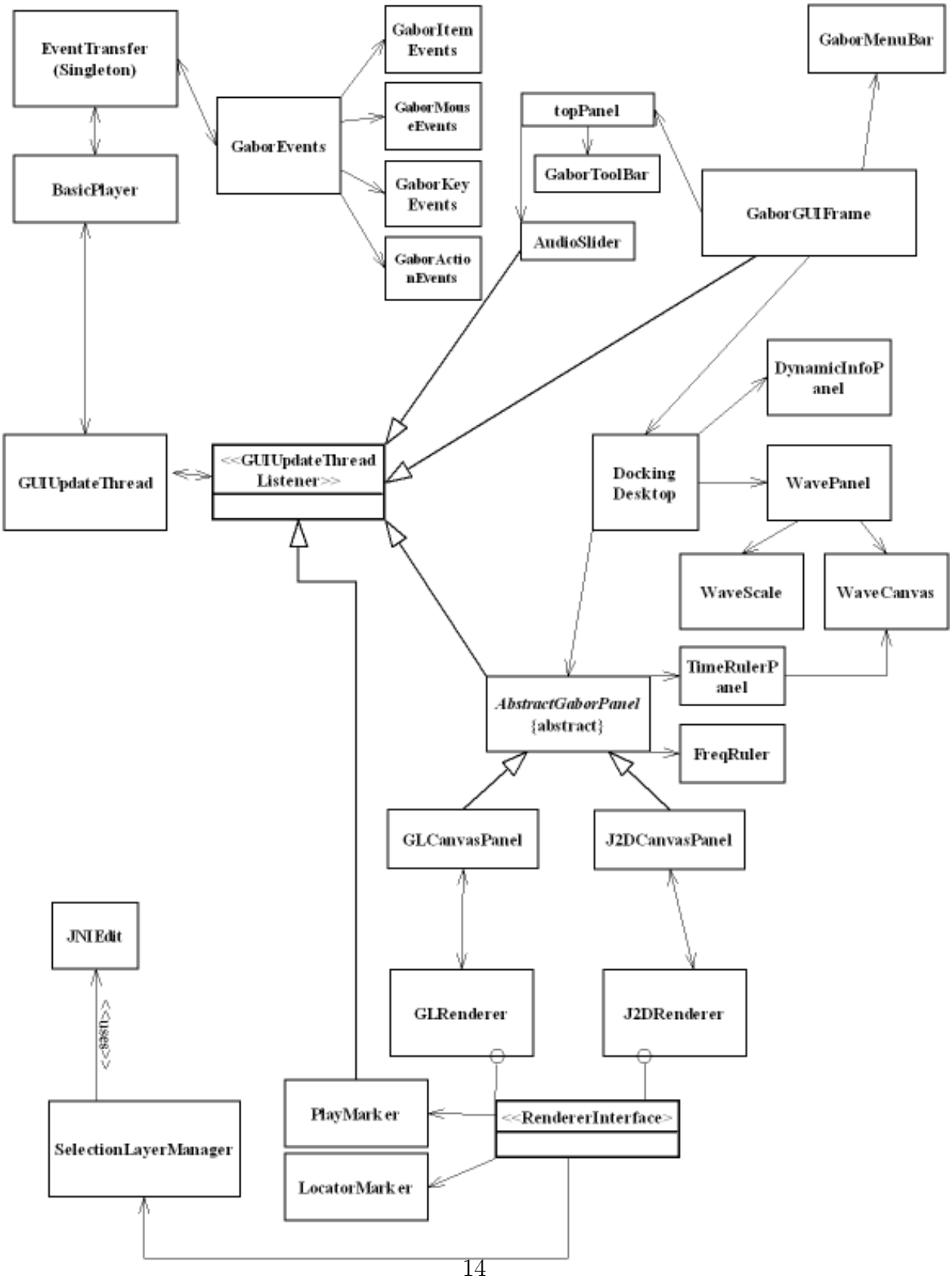


Abbildung 3.1: Klassendiagramm GaborGUI

3.2 Aufbau des Quellcodes

Im folgenden wird der Aufbau des Quellcodes anhand der verschiedenen JAVA-packages erläutert. Es werden immer nur die essentiellen Klassen behandelt, Triviales (wie FileFilter, OptionPanels u.ä.) wird nicht im Speziellen aufgeführt.

3.2.1 gui - Das Fenster zur Welt

Dieses Package beinhaltet alle Klassen, die die graphische Komponente der Benutzerschnittstelle ausmachen.

GaborGUIFrame

`GaborGUIFrame` ist die Hauptfensterklasse und enthält das Menü, die Werkzeugleiste sowie den `DockingDesktop`. Der DockingDesktop wiederum enthält drei verschiedene Elemente :

gui.swing.components.wave.WavePanel Diese Klasse ist für die Einbindung der Wellenform verantwortlich und besitzt jeweils eine Instanz der Klasse WaveScale (zur Anzeige einer Skala) sowie der Klasse `WaveCanvas`, der eigentlichen Wellenformdarstellung.

gui.swing.components.info.DynamicInfoPanel Das JPanel reagiert kontextsensitiv auf verschiedene Ereignisse und stellt dementsprechend eine oder mehrere der Klassen `ChooseHarmonicsPanel`, `OpenPolyBandwidthPanel` oder `SelectionInfoPanel` dar.

gui.AbstractGaborPanel Dem `DockingDesktop` wird (je nach gewähltem Modus) entweder ein `J2DCanvasPanel` oder ein `GLCanvasPanel` angehängt, welche beide von `AbstractGaborPanel` erben.

AbstractGaborPanel

Die beiden von `AbstractGaborPanel` erbenden Klassen sind verantwortlich für die Bereitstellung einer Umgebung zur Darstellung der Spektrogramme. Die Instanzierung dieser Klassen erfolgt durch die jeweilige Methode der Klasse `GaborPanelFactory`. Das eigentliche Zeichnen des Spektrogrammes wird von einer Implementierung des Interfaces `RendererInterface` ausgeführt.

RendererInterface

GLRenderer (bei der OpenGL-Variante) respektive J2DRenderer (für die Java2D-Implementierung) enthalten die Methoden zur Darstellung des Spektrogrammes und der zugehörigen grafischen Elemente (Loop-Lokatoren,Wiedergabeposition) sowie die Klasse (`SelectionLayerManager`)

SelectionLayerManager

Diese Klasse sorgt für einen vereinheitlichten Zugriff auf sämtliche Selektionen und deren Verwaltung. Die einzelnen Selektionsarten erben alle von der Klasse `AbstractSelectionLayer` und stellen getrennte Zeichenroutinen für die OpenGL und Java2D-Darstellung zur Verfügung. Zum Zeitpunkt der Abgabe sind lediglich die OpenGL-Darstellungs-Routinen vollständig implementiert.

3.2.2 basicplayer - Wiedergabe der Audiodateien

Dieses Modul basiert auf der BasicPlayer API V2.3[1] . Die Klasse BasicPlayer wurde so modifizert, das auch 32bit/48khz-float-Audiodateien verarbeitet werden können. Das Laden einer Audio-Datei erfolgt über die Methode `public void setDataSource(File file)`. Wird eine Datei geöffnet, die nicht nativ von Java verarbeitet werden kann, so wird die Methode `private void load32BitWave(File file)` aufgerufen. Diese liest den RIFF-Header der Wave-Datei aus und wandelt die Daten unter Verwendung der FloatConversion - Klasse in ein Format um, welches von der Soundkarte wiedergegeben werden kann.
Eine weitere Änderung stellt die Möglichkeit dar, dem BasicPlayer Loop-Punkte per `setLeftLoopLocator(float locator)` und `setRightLoopLocator(float locator)` mitzuteilen.

3.2.3 events - Interne Kommunikation

In diesem Package werden alle Event-Klassen zusammengefasst.

events.GlobalKeyListener Die Methode `public void eventDispatched(AWTEvent event)` dieses AWTEventListener ermöglicht, Key-Events global (d.h. unabhängig vom Fokus) abzufangen und zu verarbeiten. Da es jedoch bei einigen Szenarios notwendig ist, diese Funktionalität auszuschalten, wird die Methode `setActivated(boolean activated)` zur Verfügung gestellt.

[1] http://www.javazoom.net/jlgui/api.html

events.EventTransfer ist ein Singleton-Objekt und ermöglicht einen globalen Zugriff auf die wichtigsten Objekte. Die Klasse enthält Referenzen zu den Instanzen von BasicPlayer, GaborGUIFrame, PreferenceProperties sowie den Sub-Event-Klassen des GUI.

3.2.4 system - main() und co.

Die Klasse `GaborGUIMain` enthält die main()-Methode, die jedes JAVA-Programm als Einstiegspunkt benötigt.
Innerhalb dieser Klasse wird mit `System.setProperty("sun.java2d.noddraw", "true" )` auch eine „Propertie“ gesetzt, welche die Vorraussetzung für die parallele Benutzung von Java2D/Swing und JOGL unter Win32 ist. GUIUpdateThread regelt das Neuzeichnen der Anzeigekomponenten Spektrogramm und Wellenform, sowie die Synchronisation „Audiodatei - Darstellung“. Sie ist in Form eines Singleton-Elements implementiert und enthält mehrere Methoden die dem Thread mitteilen, welche Elemente neu gezeichnet werden müssen und in welcher Frequenz dies geschehen soll. Über die Methode `updateCursor()` sorgt sie für die Synchronisation der einzelnen Anzeigenelemente mit der Audiodatei bzw. bei Veränderung der Abspielposition durch den Benutzer die Synchronisation des Wiedergabemodules mit der Anzeige.

3.2.5 tools - Verschiedene Hilfsklassen

Dieses Package enthält diverse kleinere und größere Hilfsklassen, welche unter anderem der Umwandlung von Bitwerten (`LittleBitHelpers`), dem Speichern der Programmeinstellungen in eine Konfigurationsdatei (`PreferenceProperties`) oder dem Auslesen von Wave-Dateien dienen (`WaveFileTools`).

3.2.6 nativeinterface - Die Verbindung zur C-Welt

Hier werden alle Klassen zusammengefasst, welche direkt mit der externen Bibliothek kommunizieren. Allen Klassen ist gemein, das sie als Singleton-Objekte angelegt sind und somit global angesprochen werden können.

FileHandling Die Methoden
`String wavStr2trafoStr (String strWavFile, float b_crit` und
`String wavStr2pngStr (String strWavFile, float b_crit)` geben den Namen der Trafo- oder Bilddatei zurück, welche zu den Übergabewerte `strWaveFile` und `b_crit` korrenspondieren. Die Methode

`String check_png(String strWavFile,float b_crit)` prüft nach, ob schon eine Bilddatei zu diesen Parametern existiert.

JNIEdit Die bis dato einzige Methode dieser Klasse, `void jniEdit(...)`, fällt die zentrale Rolle zu, die Resynthese der Audiodatei zu veranlassen. Neben Dateiname und bCrit-Wert wird ein Array von Punkten übergeben, welche einen oder mehrere Regionen markieren. Die letzten drei Parameter stehen für die zu verwendende Operation und zwei variable Parameter. Diese Werte werden aus den drei Feldern am rechten Rand der GaborToolBar ausgelesen.

SupportFunctions Hier werden verschiedenen Methoden delegiert, welche die Interaktion mit den Daten ermöglichen, z.B. die Umrechnung von (Bild-)Vektoren in Sample- oder Frequenzwerte und zurück.

TrafoF Die Methode `float[] j_probe_b_crit()` gibt ein Array der möglichen bCrit-Werte zurück, während `float j_b_crit2b_critF(float b_crit)` zu einer Float-Zahl den nächstgelegenen zulässigen bCrit-Wert zurückgibt.

Im letzten Abschnitt dieses Kapitels wird detailliert beschrieben, welche Schritte zur Einbindung neuer native Methoden in die GUI nötig sind (Einbindung nativer Programmteile (JNI)).

3.3 Externe Bibliotheken/Klassen

Zur Erstellung des Programmes wurden einige externe Bibliotheken benutzt, welche nicht extra modifiziert wurden. Dieser Abschnitt soll einen kurzen Überblick über die Funktionsweise und den Sinn dieser Klassen geben.

3.3.1 JOGL - Java Bindings for OpenGL

Das JOGL-Projekt[2] bildet die Grundlage für die explizite Verwendung von OpenGL-Funktionen innerhalb von JAVA. Es bietet vollen Zugriff auf die API's nach OpenGL 2.0 - Spezifikationen, sowie nahezu allen herstellerabhängigen Erweiterungen (vendor extensions). Im Gegensatz zu Java3D[3] stellt JOGL einen Low-Level-Ansatz für die Verwendung von OpenGL mit JAVA dar. Da die Vorteile von Java3D entweder keinen Nutzen boten (wie der Szenengraph) oder nicht weit genug gingen (das Laden von Texturen), wurde JOGL das „tool-of-choice“ für dieses Projekt.

[2] https://jogl.dev.java.net/

[3] https://java3d.dev.java.net/

3.3.2 VLDocking - Layoutmanager

VLDocking ist ein unter der CeCILL-Lizenz[4] (eine GPL-kompatible Lizenz) zur Verfügung gestelltes Framework für JAVA. Es ermöglicht Swing-Applikationen eine vom Benutzer frei konfigurierbare Anordnung der Fenster vorzunehmen (Docking).
Der Grund für die Verwendung dieses Frameworks liegt in der (bis einschliesslich Java 1.5) existierenden „Lightweight-Heavyweight“ - Problematik von Swing in JAVA[5]. Da JOGL zum Zeitpunkt der Entwicklung des Programmes ausschließlich als Heavyweight-Komponente Hardwarebeschleunigung erfuhr, musste ein Weg gefunden werden, dieses Problem zu umgehen - VLDocking bot sich schließlich als Lösung an, da es eigene Mechanismen im Umgang mit Swing-Fenstern implementiert.

3.3.3 FloatConversion - Verabeitung von 32Bit-Float-Audiodateien

`FloatConversion` entstammt einer Diplomarbeit von Manuel René Robledo Esparza zum Thema „Java Sound”[1]. Es werden mehrere Klassen zur Verfügung gestellt, die es JAVA ermöglichen, einen 32bit-Float-Audiodaten-Stream zu dekodieren.

3.4 Einbindung nativer Programmteile (JNI)

Die Einbindung nativer Methoden in ein JAVA-Programm erfolgt in vier Schritten :

1. Innerhalb des Java-Quellcodes wird ein Methodenrumpf erstellt, welcher sowohl den Rückgabewert also auch die zu übergebenden Parameter beinhaltet. Die Methode erhält zur Kennzeichnung das Schlüsselwort `native`.
   (Beispiel: `public native float j_b_crit2b_critF(float b_crit)` in der Klasse TrafoF.java)

2. Nachdem das Projekt nun einmal kompilert hat, wechselt man in die Kommandozeile/Shell und dort in das root-Verzeichnis des Programmes. Durch Aufruf von `javah <packagename>.filename` [6] wird nun eine C-Header-Datei erzeugt.
   (Bei vorherigem Beispiel folgender Aufruf : javah nativeinterface.TrafoF)

3. Die erzeugte .h-Datei enthält die Prototypen aller als native gekennzeichneten Methoden einer Klasse. Sie wird in den C-Teil „included“, ihre Methoden müssen mit Funktionalität versehen werden (entweder ein Mapping auf vorhandene, native Funktionen oder auch ein neue Implementierung).

[4] http://www.cecill.info/

[5] http://java.sun.com/products/jfc/tsc/articles/mixing/

[6] javah ist Bestandteil des JDK

4. Ist dies geschehen, so muss eine Bibliothek erzeugt werden, welche über System.loadLibary(„NameDerLibrary“) in JAVA eingebunden werden kann. Zu beachten ist, das die Bibliothek in einem der java.library.path - Verzeichnisse liegt, so das der Linker diese auch findet.

Für eine detaillierte Referenz zum Thema „Java Native Interface“ möchte ich auf [4] verweisen[7].

[7]HTML-Version unter http://java.sun.com/docs/books/jni/ verfügbar

Kapitel 4

Designentscheidungen

Dieses Kapitel soll erläutern, welche konkreten Probleme und Herausforderungen bei der Implementierungen auftraten. Nach der Beschreibung des Problems wird (soweit vorhanden) der fertig Lösungsweg dargestellt oder gegebenfalls angedachte Wege zur Umgehung des Problems beschrieben.

4.1 Effizientes Handling großer Bilddateien

Problem : Bei der in dieser Software angestrebten Visualisierung der Audiodaten können Bilder entstehen, die extreme Dateigröße und Abmessungen besitzen. Die Ursache für letzteres ist in der Wahl der Fenstergröße (bCrit) zu suchen, die bei manchen Einstellungen Bilder mit sehr wenigen Pixeln in einer Dimension und gleichzeitig viele Pixel in der anderen Dimension entstehen lässt. Da herkömmliche Bildanzeige-Operationen (Java2D) sowohl bei der Anzeige als auch bei Skalierung/Transformation Schwierigkeiten haben, musste ein Weg gefunden werden, derartige Dateien möglichst effizient zu verarbeiten, ohne Prozessor und Speicher zu stark zu belasten.

Lösungsansatz : Die explizite Verwendung der in den meisten Rechnern vorhandenen 3D-Grafikprozessoren zur Darstellung der Bilddateien.

Ausführung : Der einfachste Weg, 3D-Hardware direkt von JAVA aus anzusprechen, führt über einer API namens „Java Bindings for OpenGL“(JOGL).Sie stellt über native Biblotheken eine Brücke nach OpenGL her, repräsentiert durch ein Objekt des Typs `GL`. Die Grafiksprache OpenGL lässt sich als eine Art „Zustandsautomat“ verstehen, d.h. das

die Verarbeitung aller Eingaben vom aktuellen Zustand abhängt [**?**]. Eine weitere wichtige Eigenschaft von OpenGL ist die Auslegung als „Single-Threaded-Library“, sämtliche Operation müssen innerhalb der display()- Routine aufgerufen werden.
Wie schon im vorherigen Kapitel beschrieben erfolgt die OpenGL-Darstellung in der Klasse `GLRenderer` respektive deren Methode display(). Die Methode läuft solange im „Leerlauf“ bis der Benutzer das eine Datei zur Darstellung auswählt. Durch den Aufruf von `loadTexture(GL gl)` der Subklasse `TextureImage` wird das Ladevorgang initialisiert. Sie prüft dessen Dimensionen und benutzt die Klasse `LargeTextureTiler` zur Aufbereitung des Bildes für die Verwendung als Textur. Diese Aufbereitung ist notwendig, da die meisten OpenGL-Treiber eine Texturen-Auflösung von maximal 4096*4096 Pixeln anbieten. Das Bild in mehrere Einzelbilder aufgeteilt, die resultierenden Bilder werden als Byte-Array in den Speicher gelegt der Grafikkarte als Textur bekannt gemacht.
Alle weiteren Bildoperationen finden jetzt direkt auf der Grafikkarte statt, nur die Anteile des Bildes, welche nicht auf dem Bildschirm zu sehen sind, werden in den Hauptspeicher ausgelagert.

4.2 Behandlung von Mouse-Events in OpenGL

Problem : Die Umsetzung von Mouse-Events, welche direkt mit der Grafikausgabe interagieren, wird durch zwei Faktoren erschwert :

1. Um OpenGL mitzuteilen, welches Objekt der Benutzer konkret manipulieren will, müssen Bildschirm- nach Weltkoordinaten umgerechnet und anschliessend das in Frage kommende Objekt bestimmt werden.

2. Diese Umrechnung kann nur zu einem Zeitpunkt geschehen, an welchem der GL-Kontext gültig (valid) ist.

Lösungsansatz : Sämtliche Maus-Ereignisse müssen innerhalb der display()-Methode verarbeitet werden.

Ausführung : Zunächst wird jedes relevante Maus-Ereignis (Klick + Drag) in der Klasse `MouseQueue` gespeichert. Neben den Koordinaten werden auch die Informationen übergeben, welche Taste gedrückt wird und ob es ein einfacher Klick, ein Doppelklick oder ein Drag-Aktion ist. Beim nächsten Aufruf der display()-Methode wird diese Warteschlange dann wie folgt verarbeitet :

- Umrechnung der Mauskoordinaten in OpenGL-Weltkoordinaten via `gluUnProject()`

- Löschen der verarbeitenden Events und Speichern in der `events_processed` - Warteschlange
- Vergleich der errechneten Weltkoordinaten mit den aktuellen Objekten (Selection-Points,Loopmarker), bei Übereinstimmung :
- Manipulation des Objektes durch das Maus-Ereignis (Selektion/Verschiebung von Punkten, Setzen der Lokatoren...)
- Löschen der Warteschlange

4.3 Wiedergabe und Verarbeitung von 32bit-Float-Waves mit JAVA

Problem : JAVA unterstüzt standardmäßig weder bei Verarbeitung noch bei der Wiedergabe Audiodateien im 32bit-Float-Format.

Lösungsansatz : Manuelles Handling der nicht von JAVA automatisch erkannten Audiodateien.

Ausführung : Erster Schritt ist die Erkennung, ob eine Datei von JAVA gelesen werden kann. Dieses geschieht in der Methode `setDataSource(File f)` welche zunächst Versucht, die Datei mit nativen Java-Methoden zu laden. Ist dies nicht Möglich, so wird eine `UnsupportedAudioFileException` geworfen. In deren catch()-Block wird nun die Methode `getDataChunkOffset()` der Hilfsklasse `WaveFileTools` aufgerufen, welche den RIFF-Header der Wave-Datei überprüft und die Position des DATA-chunks zurückgibt[1]. Mithilfe der FloatConversion-Klasse (siehe Kapitel 3.3.3) wandelt man nun den 32bit-Float-Datenstrom in einen 16bit-Integer-Datenstrom um. Dieser kann nun wie ein normaler `AudioInputStream` vom Wiedergabemodul (BasicPlayer) benutzt werden.

4.4 CPU-schonende Anzeige einer Audio-Wellenform

Problem : Die Darstellung großer Audiodateien als Wellenform belastet die CPU durch eine hohe Zahl an Rechenoperationen.

[1] zukünftige Versionen dieser Methode sollen auch das Format der Datei durch das Auslesen des RIFF-Headers bestimmen können

Lösungsansatz : Minimierung der CPU-Last durch die Vorberechnung mehrerer Ebenen, auf deren Grundlage die Wellenform gezeichnet werden soll.

Ausführung : Beim Laden einer Audiodatei wird zunächst eine Methode aufgerufen, welche in Abhängigkeit der maximalen Bildschirmbreite verschiedene Layer einer Wellenform berechnet und in einem Vector speichert. Bei jedem Aufruf von paintComponent() wird nun durch die Methode `int chooseSampleArray(int samples_per_pixel)` bestimmt, welches der optimale Layer für die Darstellung dieses Bildausschnitts ist. Nun wird jedem darzustellendem Bildpunkt eine Menge von Werten dieses Arrays zugewiesen und das Minimum/Maximum für jedes horizontale Pixel bestimmt. Zwischen diesen beiden Punkten wird abschliessend eine Linie gezeichnet, welche die Amplitude der Wellenform an diesem Zeitpunkt entspricht.
Da dieser Weg zum Zeitpunkt der Abgabe nicht zufriedenstellend funktioniert, wird noch auf eine einfachere Methode zurückgegriffen, welche (je nach Größe des darzustellende Bereiches) erhebliche CPU-Last verursacht.

Kapitel 5

Ausblick

Zum Abschluss dieser Arbeit soll in diesem Kapitel das Erreichte nocheinmal zusammengefasst sowie ein Ausblick auf mögliche Erweiterungen des Programmes gegebenen werden. Zuletzt soll auf die noch zu lösenden Probleme und Fehler eingegangen werden.

5.1 GaborGUI: Gegenwart - Zukunft

Gegenwart

Die vorliegende Software kann die wichtigsten Ansprüche an die von ihr geforderten Fähigkeiten erfüllen.
Sie ermöglicht ein stabiles Arbeiten auf einem Großteil der heutigen Rechner, nur bei einigen Konfigurationen (speziell unter Linux, siehe nächster Abschnitt) kommt es zu größeren Schwierigkeiten bei der Benutzung des Programmes. Sind die Systemvorraussetzungen erfüllt, so ist die Installation des Programmes denkbar einfach - nach dem Entpacken des Archives auf die Festplatte reicht das simple Aufrufen einer Batch-/Skriptdatei zur Ausführung des Programmes. Daraus resultiert auch die in der Aufgabenstellung geforderte Portabilität.
Durch die Verwendung von OpenGL als (Haupt-)Visualisierungsmodul ist auch für die Anforderung das Programm möglichst ressourcensparend zu verwirklichen ein gute Grundlage gelegt. Eine Modularisierung des Programmes konnte zumindest teilweise umgesetzt werden. Durch die Verwendung abstrakter Klassen und Interfaces ist es unter relativ geringem Aufwand möglich, neue Methoden der Visualisierung zu implementieren oder vorhandene anzupassen.
Die eigentlichen Editier-Funktionen sind noch als eher rudimentär zu Betrachten, erfüllen jedoch ihre Zwecke zuverlässig und intuitiv.

Zukunft

Für die Zukunft des Programmes sind mehrere Punkte von herrausragender Bedeutung, welche den Nutzwert erheblich steigern können. Im einzelnen sind dies :

- Simultane Darstellung/Verwendung mehrerer Audiodateien. Die Implementierung dieser Fähigkeit wird mit Verwendung des derzeit im Beta-Stadiums befindlichen JDK 6 (Mustang) erheblich vereinfacht. Dieses bietet eine sogenannte Java2D/JOGL Interoperability - Bridge an, die das Zusammenspiel dieser beiden Systeme erheblich verbessert.
- Verwendung komprimierter Audiodateien der verschiedensten Formate. Die bisherige Beschränkung auf 48khz/32bit-Float Dateien limitiert die einfache Benutzung doch recht stark. Am einfachsten erreicht werden kann dies durch eine Kombination aus JAVA-Code (das Einlesen der Dateien) und C-Code (Konvertierung der Dateien mithilfe der libsndfile-Bibliothek).
- Implementierung fortschrittlicher Selektionsfunktionen. Die bisherigen Methoden decken nur einen kleinen Teil der möglichen Anwendungen ab. Neben einer „Zauberstab“-Funktion, welche automatische Region erkennt und markiert, wären „Hit“-Selektionen und andere Bezièr-basierte Selektionen wünscheswert.
- „Übersichts“-Darstellung der aktuellen Audiodatei (würde speziell das Handling großer Dateien vereinfachen).
- Umsetzung des Programmes als JAVA-Applet zur Ausführung in Internetbrowsern.

Neben diesen Punkten existieren natürlich noch eine Vielzahl weiterer Möglichkeiten zur Verbesserung der Benutzbarkeit, unter anderem die Einführung von „Fortschrittsanzeigen“ bei Aktionen wie der Berechnung von trafo-Dateien oder auch ein Kontextmenü zur Auswahl der Werkzeuge.

5.2 Bekannte Fehler und Probleme

Zum Zeitpunkt der Abgabe existieren in der Anwendungen noch folgende Fehler und Probleme :

- Bei Benutzung des Programmes mit ATI-Grafikkarten unter Linux wird das Programm nach dem zweiten Neuladen einer Datei terminiert. Dieser Effekt tritt bei der Verwendung des Open-Source-Treibers auf und ist auf eine fehlerhafte Implementierung dessen OpenGL-Parts zurückzuführen.

- In einigen Fällen ist die Loop-Wiedergabe ungenau, abhängig von der Position der Locatoren.
- Die Anzeige der aktuellen Position bei der Wiedergabe verläuft nicht flüssig.
- Bei der Verwendung des OpenGL-Anzeigemodules kommt es bei einer Umordnung des Spektrogramm-Fensters zu Darstellungsproblemen, es werden keinerlei Texturen (=Bild) mehr angezeigt. Durch erneutes Laden des Bildes wird das Problem temporär behoben.
- Die Implementierung des Java2D-Parts ist bis dato sehr rudimentär und eher als „Proof-Of-Concept“ anzusehen. Sie ermöglicht zwar die Anzeige des Bildes und die Darstellung eines PlayMarker, hat jedoch mit erheblichen Performance-Problemen zu kämpfen und weisst keinerlei Selektions- oder Loop-Funktionen auf. Von der Benutzung dieser Möglichkeit wir im Moment abgeraten.

Anhang A

Anhang

Der Anhang enthält eine Übersicht der Tastatur- und Mauskommandos, eine Beschreibung des CD-Inhaltes sowie Angaben zu genutzten Quellen

A.1 Tastatur- und Mauskommandos

Common Actions

SPACE - toggle play
LEFT,UP,DOWN,RIGHT - KEYS - pan the image (needs focus on GaborPanel)
PAGE UP, PAGE DOWN - zoom the image (needs focus on GaborPanel)
CTRL + O - load new file
DEL - delete all marked points
F5 - choose „arrow-tool“
F6 - choose „rectangle-tool“
F7 - choose „open-poly-tool“
F - toggle followmarker
L - toggle loop
MOUSEWHEEL-DRAGGING - pan the image
MOUSEWHEEL-ROTATE - zoom the image
ESC - clear selection

Arrow Tool

CLICK ON POINT - mark point and set current
CLICK ON POINT + CTRL - mark point and set current. keep old marked points
DRAG POINT - translate the point
DRAG POINT + CTRL - translate all marked points
DOUBLECLICK ON POINT - mark the whole layer

Rectangle Tool

CLICK + DRAG - create new rectangle
CLICK + DRAG + SHIFT - create new rectangle, keep old ones
CLICK + DRAG + CTRL - mark point and set current. keep old marked points,drag

Open-poly Tool

CLICK - add poly point to current polygon
CLICK + SHIFT - add poly to new polygon
CLICK + DRAG + CTRL - mark point and set current. keep old marked points,drag

A.2 Ordnerstruktur des Datenträgers

Die beiliegende Compact-Disc enthält folgende Verzeichnisse :

source Der Quellcode des Programmes

win32 FULL und LITE - Versionen des Programmes für die Win32-Plattform

linux FULL und LITE - Versionen des Programmes für Linux

wave Einige Beispieldateien zur Verwendung des Programmes. Müssen zur Benutzung auf einen beschreibbaren Datenträger kopiert werden.

javadoc Die automatisch generierte Java-Dokumentation

Literaturverzeichnis

[1] Manuel René Robledo Esparza. Javasound als plattform für die entwicklung studiotauglicher audioapplikationen. Master's thesis, Fachhochschule Stuttgart, 2004.

[2] R. Lienhart Gregor van den Boogaart. Audio brush: Editing audio in the spectrogram. Technical Report 510, Universität Augsburg Institut für Informatik, June 2005.

[3] R. Lienhart Gregor van den Boogaart. Fast gabor transformation for processing high quality audio. Technical Report 521, Universität Augsburg Institut für Informatik, October 2005.

[4] Sheng Liang. *The Java Native Interface - Programmer's Guide and Specification.* Addison-Wesley, 1999.

Abbildungsverzeichnis